In the Specification

Please amend the specification of this application as follows:

Rewrite the paragraph at page 1, lines 12 to 23 as follows:

A2

--Traditionally, Very Long Instruction ~~Word, VLIW~~ Word (VLIW) processors have been defined by the following set of attributes. ~~The~~ These processors have the ability to specify multiple, independent operations in each ~~instruction.~~ instruction (a MultiOp ~~instruction.)~~ instruction). VLIW architectures are horizontal machines, with each wide instruction-word or *MultiOp*, consisting of several operations or , *Ops*. All Ops in a MultiOp are issued in the same execution schedule. Programs that assume specific non-unit latencies for the operations and which, in fact, are only correct when those assumptions are true. The requirement for static, compile-time operation scheduling ~~taking~~ takes into account operation latencies and resource availability. Consequently, ~~the requirement that~~ the hardware must conform exactly to the assumptions built into the program with regards to the number of functional units and the operation latencies. ~~The absence of~~ VLIW processors typically lack any interlock hardware, despite the fact that multiple, pipelined operations are being issued every cycle.--

Rewrite the paragraph at page 1, lines 24 to 30 as follows:

A3

--The original attraction of this style of architecture is its ability to exploit large amounts of instruction-level parallelism (ILP) with relatively simple and inexpensive control hardware. Whereas a number of VLIW products have been built which are capable of issuing six or more operations per cycle ~~[4, 5, 3]~~, it has just not proven feasible to build superscalar products with this level of ILP ~~[18, 2, 14, 8, 7, 6]~~. Furthermore, the complete exposure to the compiler of the available hardware resources and the exact operation latencies permits highly optimized) schedules.--

Rewrite the paragraph at page 2, line 26 to page 3, line 6 as follows:

A4

--A different view of VLIW is as an architecture, i.e., a contractual interface between the class of programs that are written for the architecture and the set of processor Implementations of that architecture. The usual view is that this contract is concerned with the instruction format and the interpretation of the bits that constitute an ~~Instruction~~ instruction. But the contract goes further and it is these aspects of the contract that are of primary importance in this patent. First, via its MultiOp capability, a VLIW architecture specifies a set of operations that are guaranteed to be mutually independent (and which, therefore, may be issued simultaneously without any checks being made by the issue hardware).--

Rewrite the paragraph at page 3, line 30 to page 5, line 13 as follows:

A5

--Very Long Instruction ~~Word, VLIW,~~ Word (VLIW) processors are viewed as an attractive way of achieving instruction-level parallelism because of their ability to issue multiple operations per cycle with relatively simple control logic. Although VLIW architectures offer the advantages of simplicity of design and high issue rates, a major impediment to the use of VLIW and other novel ILP architectures is that they are not compatible with the existing software base. Lack of object code compatibility in VLIW architectures across processors having different hardware latencies and varying levels of parallelism is a severe limit to their adoption as a general purpose computing paradigm. This means that an installed software base of binaries cannot be built around a family of generations. The economic implications of this problem are enormous, and an efficient solution is necessary if VLIW

A5

architectures are to succeed. Two classes of approaches to this problem have been reported in the literature: hardware approaches and software approaches. Although these techniques may provide compatibility, they do so at the expense of hardware complexity that can potentially impact cycle time. A typical software approach is to statically recompile the VLIW program from the object file. The approach generates multiple executables, which poses difficulties for commercial copy protection and system administration. For example, if a first generation of a machine has certain latencies involved with each functional unit and the second generation VLIW machine has different latencies involved with those same functional units, the old VLIW schedule cannot be guaranteed to execute properly on the second generation machine due to the flow dependence between the different operations. The same type of problem results if the second generation machine includes an additional functional unit. Even if the latencies remained the same, the code scheduled for this new machine would not execute correctly on the older machines because the scheduler has moved operations in order to take advantage of the additional ~~multiplier~~ functional unit. There is no trivial way to adapt this schedule to the older machines. This is the case of downward incompatibility between generations. In this situation, if different generations of the machines share binaries, compatibility requires either a mechanism to adjust the schedule or a different set of binaries for each generation. IBM describes hardware features for an ILP machine called DAISY (Dynamically Architected Instruction Set from Yorktown). DAISY is specifically intended to emulate existing architectures, so that all existing software for an old architecture (including operating system kernel code) runs without changes on the VLIW architecture. Each time a new fragment of code is executed for the first time, the code is translated to VLIW primitives, parallelized and saved in a portion of main memory not

A5

visible to the old architecture, by a Virtual Machine Monitor (software) residing in read only memory. Subsequent executions of the same fragment do not require a translation (unless cast out). A limitation of the hardware approaches is that the scope for scheduling is limited to the window of Ops seen at run-time, hence available ILP is relatively less than what can be exploited by a compiler. These schemes may also result in cycle time stretch, a phenomenon ~~due to which many are~~ of concern when considering the VLIW paradigm over superscalar for future generation machines.--

Rewrite the paragraph at page 6, lines 6 to 24 as follows:

A6

--A subpipelined translation embodiment providing binary compatibility between current and future generations of DSPs is disclosed. When a fetch packet is retrieved from memory, the entire fetch packet is assigned an operating mode (base instruction set or migrant instruction set) according to the execution mode at the time the request was made to the instruction memory for the fetch packet. The fetch packets from the instruction memory are parsed into execute packets and sorted by execution unit (dispatched) in a datapath shared by both execution ~~modes(base~~ modes (base and migrant). Because the fetch packet syntax and execution unit encoding is different between the migrant and base architecture in this case, the two execution modes have separate control logic. Instructions from the dispatch datapath are decoded by either base architecture decode logic or the migrant architecture decode logic, depending on the execution mode bound to the parent fetch packet of the instructions being decoded. Code processed by the migrant and base decode pipelines produces machine words that control the register files and the execution hardware functional units. These machine words are selected with a multiplexer. The choice of the eventual machine word from the multiplexer is governed by the operating mode bound to the fetch

A6

packet that produced the machine word and sequencing logic for subpipelined execution. The selected machine word controls a global register file, which supplies operands to all hardware execution units and accepts results of all hardware execution units.--

Rewrite the paragraph at page 7, lines 7 to 14 as follows:

A7

--Subpipelined execution is a hardware-efficient method for executing code from a migrant exposed-pipeline architecture. In this method, the base architecture is designed with a fixed ~~multiple(call~~ multiple (call this *S*) of the instruction latencies of the desired migrant architecture. With this relationship between the migrant and base architectures, code from the migrant architecture can be executed on the base architecture by delaying the issue of migrant instructions by *S-1* clock cycles. In addition to providing the subpipelined execution mode, facilities are provided to change between base and migrant instruction sets with low overhead.--

Rewrite the paragraph at page 8, lines 20 to 28 as follows:

A8

--Although the previously ~~discussed(in~~ discussed (in the Background of Invention) IBM solutions and other HP and North Carolina State University solutions on supporting compatibility between generations of VLIW processors may result in VLIW base architectures that can support migrant architectures, these solutions are not appropriate solutions for DSPs. In particular, dynamic translation approaches as described by the IBM and NCSU works fail to provide the run-time predictability required for real-time DSP applications. The method described in the HP work will have complex but deterministic run-time, but substantial hardware costs from the delayed -issue instruction buffer, delay register file and copyback unit.--

Rewrite the paragraph at page 9, lines 15 to 19 as follows:

A9

--The fetch packets from the instruction memory are parsed into execute packets that can be scheduled to execute simultaneously and sorted by execution unit (dispatched) in a datapath shared by both execution ~~modes(base~~ modes (base and migrant). Because the fetch packet syntax and execution unit encoding is different between the migrant and base architecture in this case, the two execution modes have separate control logic.--

Rewrite the paragraph at page 9, lines 20 to 31 as follows:

A10

--Instructions from the dispatch datapath are decoded by either base architecture decode logic or the migrant architecture decode logic, depending on the execution mode bound to the parent fetch packet of the instructions being decoded. In the case of exposed-pipeline VLIW instruction sets, the decode logic for the base and migrant architectures primarily translates opcodes to the control signals required to execute the specified instructions on the execution hardware functional units. Because of the relationship defined in this invention with regard to the latencies of the migrant and base instruction ~~sets(the~~ sets (the base operations have twice the latency of the migrant operations) and the exposed pipeline characteristics of these instruction sets, instruction decoding techniques that need knowledge of the pipeline depth, instruction graduation and instruction dependencies are not required. This results in reduced hardware and complexity in the instruction decode logic.--